ELSEVIER

Artificial Intelligence 137 (2002) 275–278

Artificial Intelligence

www.elsevier.com/locate/artint

Forthcoming Papers

Knowledge Representation and Logic Programming (Special Issue edited by Michael Gelfond and Nicola Leone)

M. Gelfond and N. Leone, Logic programming and knowledge representation—The A-Prolog perspective

In this paper we give a short introduction to logic programming approach to knowledge representation and reasoning. The intention is to help the reader to develop a 'feel' for the field's history and some of its recent developments. The discussion is mainly limited to logic programs under the answer set semantics. For understanding of approaches to logic programming build on well-founded semantics, general theories of argumentation, abductive reasoning, etc., the reader is referred to other publications. © 2002 Published by Elsevier Science B.V.

V. Lifschitz, Answer set programming and plan generation

The idea of answer set programming is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use an answer set solver, such as SMODELS or DLV, to find an answer set for this program. Applications of this method to planning are related to the line of research on the frame problem that started with the invention of formal nonmonotonic reasoning in 1980. © 2002 Published by Elsevier Science B.V.

G. Gottlob, F. Scarcello and M. Sideri, Fixed-parameter complexity in AI and nonmonotonic reasoning

Many relevant intractable problems become tractable if some problem parameter is fixed. However, various problems exhibit very different computational properties, depending on how the runtime required for solving them is related to the fixed parameter chosen. The theory of parameterized complexity deals with such issues, and provides general techniques for identifying fixed-parameter tractable and fixed-parameter intractable problems.

We study the parameterized complexity of various problems in AI and nonmonotonic reasoning. We show that a number of relevant parameterized problems in these areas are fixed-parameter tractable. Among these problems are constraint satisfaction problems with bounded treewidth and fixed domain, restricted forms of conjunctive database queries, restricted satisfiability problems, propositional logic programming under the stable model semantics where the parameter is the dimension of a feedback vertex set of the program's dependency graph, and circumscriptive inference from a positive k-CNF restricted to models of bounded size. We also show that circumscriptive inference from a general propositional theory, when the attention is restricted to models of bounded size, is fixed-parameter intractable and is actually complete for a novel fixed-parameter complexity class. © 2002 Published by Elsevier Science B.V.

0004-3702/2002 Published by Elsevier Science B.V.
PII: S0004-3702(02)00216-3

J.J. Alferes, L.M. Pereira, H. Przymusinska and T.C. Przymusinski, LUPS—A language for updating logic programs

Most of the work conducted so far in the field of logic programming has focused on representing static knowledge, i.e., knowledge that does not evolve with time. To overcome this limitation, in a recent paper, the authors introduced *dynamic logic programming*. There, they studied and defined the declarative and operational semantics of sequences of logic programs (or dynamic logic programs). Each program in the sequence contains knowledge about some given state, where different states may, for example, represent different time periods or different sets of priorities.

But how, in concrete situations, is a sequence of logic programs built? For instance, in the domain of actions, what are the appropriate sequences of programs that represent the performed actions and their effects? Whereas dynamic logic programming provides a way for, given the sequence, determining what should follow, it does not provide a good practical language for the specification of the sequence of updates which may be conditional on the intervening states.

Here we define the language LUPS—"Language for dynamic updates"—designed for specifying changes to logic programs. Given an initial knowledge base (as a logic program) LUPS provides a way for sequentially updating it. The declarative meaning of a sequence of sets of update actions in LUPS is defined by the semantics of the dynamic logic program generated by those actions. Additionally, we provide a translation of the sequence of update statements sets into a single logic program written in a meta-language, in such a way that the stable models of the resulting program correspond to the previously defined declarative semantics. Finally, we exhibit the usage of LUPS in several application domains.

B. Cui and T. Swift, Preference logic grammars: Fixed point semantics and application to data standardization

The addition of preferences to normal logic programs is a convenient way to represent many aspects of default reasoning. If the derivation of an atom A_1 is preferred to that of an atom A_2, a preference rule can be defined so that A_2 is derived only if A_1 is not. Although such situations can be modelled directly using default negation, it is often easier to define preference rules than it is to add negation to the bodies of rules. As first noted by Govindarajan et al. [Proc. Internat. Conf. on Logic Programming, 1995, pp. 731–746], for certain grammars, it may be easier to disambiguate parses using preferences than by enforcing disambiguation in the grammar rules themselves. In this paper we define a general fixed-point semantics for preference logic programs based on an embedding into the well-founded semantics, and discuss its features and relation to previous preference logic semantics. We then study how preference logic grammars are used in *data standardization*, the commercially important process of extracting useful information from poorly structured textual data. This process includes correcting misspellings and truncations that occur in data, extraction of relevant information via parsing, and correcting inconsistencies in the extracted information. The declarativity of Prolog offers natural advantages for data standardization, and a commercial standardizer has been implemented using Prolog. However, we show that the use of preference logic grammars allow construction of a much more powerful and declarative commercial standardizer, and discuss in detail how the use of the non-monotonic construct of preferences leads to improved commercial software.

V. Marek, I. Pivkina and M. Truszczyński, Annotated revision programs

Revision programming is a formalism to describe and enforce updates of belief sets and databases. That formalism was extended by Fitting who assigned annotations to revision atoms. Annotations provide a way to quantify the confidence (probability) that a revision atom holds. The main goal of

our paper is to reexamine the work of Fitting, argue that his semantics does not always provide results consistent with intuition, and to propose an alternative treatment of annotated revision programs. Our approach differs from that proposed by Fitting in two key aspects: we change the notion of a model of a program and we change the notion of a justified revision. We show that under this new approach fundamental properties of justified revisions of standard revision programs extend to the annotated case.

P. Simons, I. Niemelä and T. Soininen, Extending and implementing the stable model semantics

A novel logic program like language, weight constraint rules, is developed for answer set programming purposes. It generalizes normal logic programs by allowing weight constraints in place of literals to represent, e.g., cardinality and resource constraints and by providing optimization capabilities. A declarative semantics is developed which extends the stable model semantics of normal programs. The computational complexity of the language is shown to be similar to that of normal programs under the stable model semantics. A simple embedding of general weight constraint rules to a small subclass of the language called basic constraint rules is devised. An implementation of the language, the SMODELS system, is developed based on this embedding. It uses a two level architecture consisting of a front-end and a kernel language implementation. The front-end allows restricted use of variables and functions and compiles general weight constraint rules to basic constraint rules. A major part of the work is the development of an efficient search procedure for computing stable models for this kernel language. The procedure is compared with and empirically tested against satisfiability checkers and an implementation of the stable model semantics. It offers a competitive implementation of the stable model semantics for normal programs and attractive performance for problems where the new types of rules provide a compact representation.

R. Ben-Eliyahu-Zohary, Yet some more complexity results for default logic

Y. Lebbah and O. Lhomme, Accelerating filtering techniques for numeric CSPs

C. Dixon, M. Fisher and A. Bolotov, Clausal resolution in a logic of rational agency

S. Bistarelli, P. Codognet and F. Rossi, Abstracting soft constraints: Framework, properties, examples

R. Greiner, A.J. Grove and D. Roth, Learning cost-sensitive active classifiers

F. Bacchus, X. Chen, P. van Beek and T. Walsh, Binary vs. non-binary constraints

C. Bettini, X.S. Wang and S. Jajodia, Solving multi-granularity temporal constraint networks

N. Jussien and O. Lhomme, Local search with constraint propagation and conflict-based heuristics

K. Knight and D. Marcu, Summarization beyond sentence extraction: A probabilistic approach to sentence compression

M. Broxvall, P. Jonsson and J. Renz, Disjunctions, independence, refinements

F. Lin and J.-H. You, Abduction in logic programming: A new definition and an abductive procedure based on rewriting

I. Navarrete, A. Sattar, R. Wetprasit and R. Marin, On point-duration networks for temporal reasoning

M. Ying and H. Wang, Lattice-theoretic models of conjectures, hypotheses and consequences (Research Note)